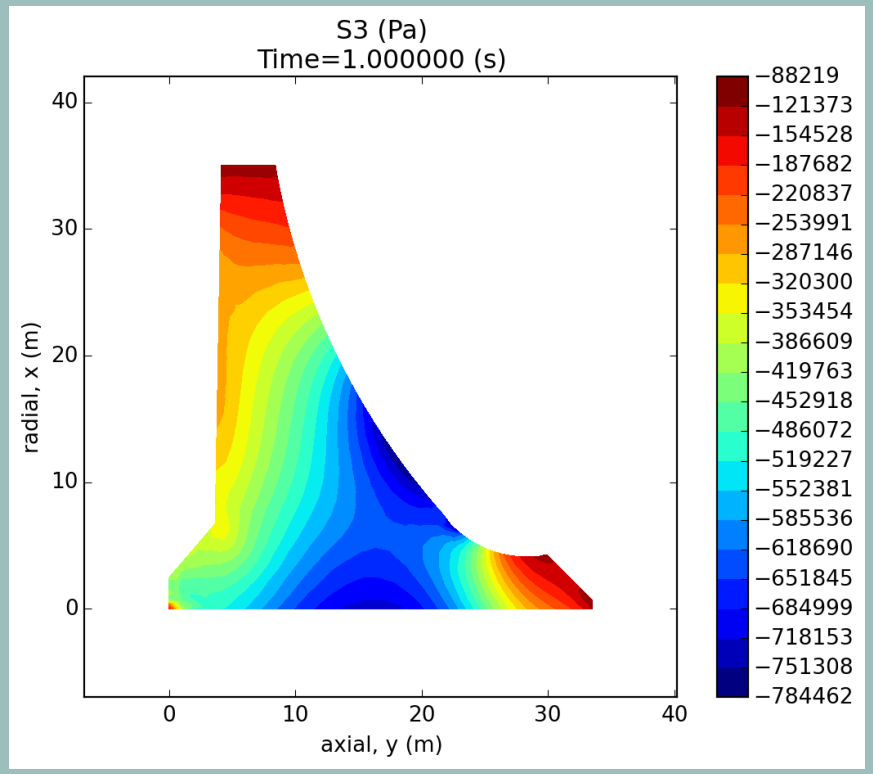
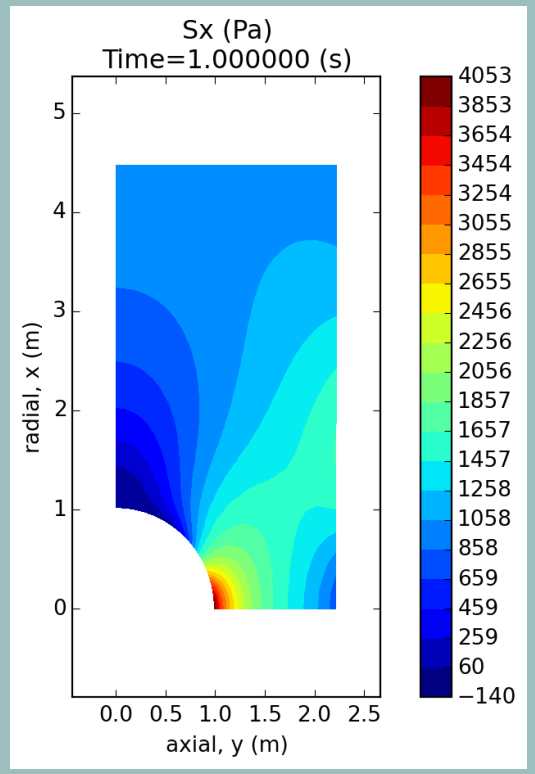
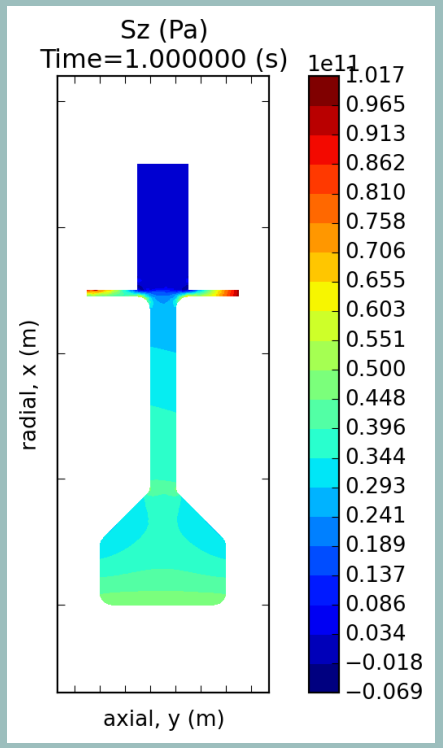






PYTHON API TO RUN CALCULIX: PYCALCULIX

2014-12-22

Justin Black

Mechanical Engineer

Justin.a.black@gmail.com

www.justinablack.com

WHAT AND WHY

What:

I created an API in python to build, solve and analyze mechanical engineering Finite Element Analysis models of parts

Forces, displacements, gravity etc can be applied to a part and displacements and stresses can be displayed and queried

Why?

The existing free tools are very capable but not very automatable or user friendly

To make a model currently, you have to learn 4 programs, I reduced that to one

FINITE ELEMENT ANALYSIS + PYCALCULIX CAPABILITIES

Mechanical engineers can analyze parts using finite element anlaysis where parts are cut into tiny square or triangular elements, and deflections stresses etc are calculated in each element and on the corner nodes. This method and similar methods are used in industry to analyze both solid metal parts and even analyze fluid flows.

In my case I limited my API to relatively simply types of 2D problems:

1. Plane Stress: 2d parts that are very thin, example: pulling on a plate
2. Plane Strain: 2d parts that are very thick, example: a dam
3. Axisymetric: a 3d part which is a 2d area revolved around an axis: jet engine case or rotor

WORKFLOW

Original

1. Make part file (1 CAD Program)
2. Mesh part in program (2 GMSH)
3. Edit mesh file to remove junk (3 TXT)
4. Write solver .inp file (3 TXT)
5. Solve file (4 Calculix CCX)
6. Look at results in gui (5 Calculix CGX)

Pycalculix

1. Make geometry
2. Apply Loads + Constraints
3. Call mesh method*
4. Make solver instance + solve**
5. Look at results

*GMSH used in the background
**Calculix CCX used in the background

New Pycalculix workflow allows for more streamlined model building and solution.
Manual editing of files eliminated. Original workflow used 5 programs.
All aspects of the workflow are accessed in one python program, Pycalculix.

QUICK START EXAMPLES IN GITHUB

HTTPS://GITHUB.COM/SPACETHER/PYCALCULIX

File	Problem Description	Element/Problem Type
dam.py	Plane strain analysis of pure concrete dam	Plane strain
hole_model.py	Plane stress analysis of hole in plate, quarter symmetry	Plane stress
compr_rotor_stage.py	Axisymmetric analysis of jet engine compressor rotor with blade (blisk)	Axisymmetric
hole_kt.py	Design study varying size of hole in plate, verifies the tension Kts from Calculix are consistent with textbook answers	Plane stress design study

PYCALCULIX REQUIREMENTS, PG1

Pycalculix.py file must currently be in the same folder as the model you are running

Python 3+* must be installed

CCX must be installed

Matplotlib* must be installed (used to make plots of parts + results)

Numpy* must be installed (this is used to find principal stresses)

*Python, Numpy, and Matplotlib are all installed in the Anaconda distribution: http://continuum.io/downloads#py34

PYCALCULIX REQUIREMENTS, PG2

I've hard-coded in the paths to:
GMSH, CCX, and CGX
In pycalculix.py:

```
83  _ccx = r'C:\Program Files (x86)\bConverged\CalculiX\ccx\ccx.exe'
84  _cgx = r'C:\Program Files (x86)\bConverged\CalculiX\cgx\cgx.exe'
85  _gmsh = r'C:\Program Files (x86)\gmsh-2.8.5\gmsh.exe'
```

One option would be to require the user to pass the paths when instantiating the FEA model [DONE]

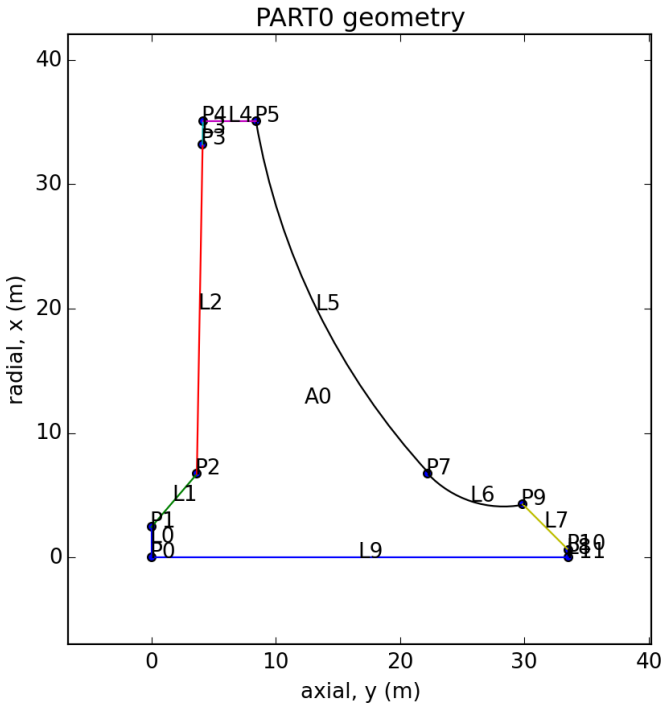
```
from pycalculix import FeaModel

# Vertical hole in plate model, make model
ccx = r'C:\Program Files (x86)\bConverged\CalculiX\ccx\ccx.exe'
cgx = r'C:\Program Files (x86)\bConverged\CalculiX\cgx\cgx.exe'
gmsh = r'C:\Program Files (x86)\gmsh-2.8.5\gmsh.exe'
proj_name = 'hole_model'
a = FeaModel(proj_name, ccx, cgx, gmsh)
```

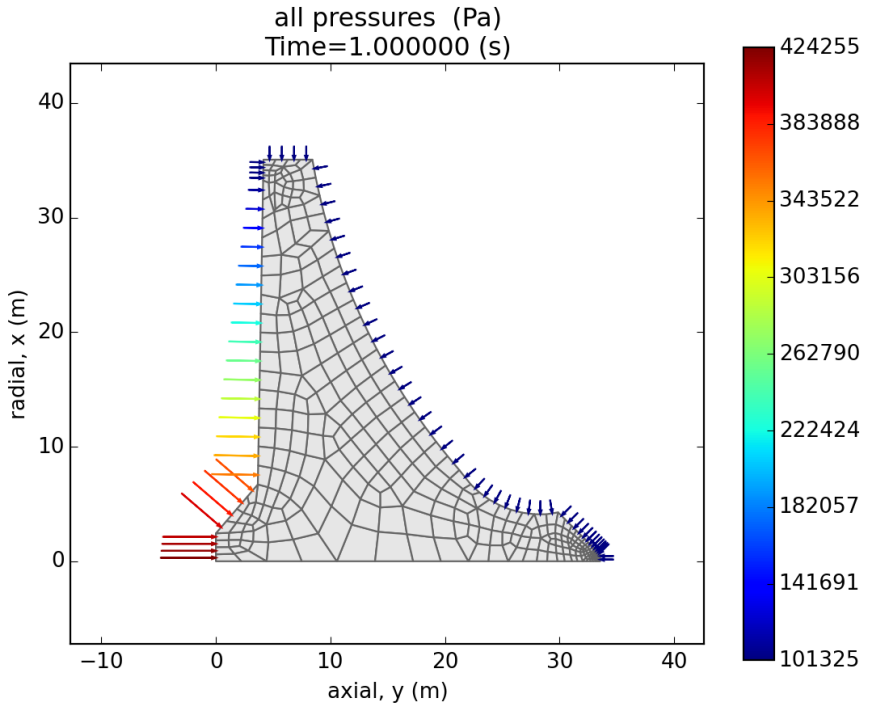
Another would be to distribute the programs with my library, and install my library with pipi (python package manager). **I'd prefer this because it would be easiest for python users.**

File: dam.py

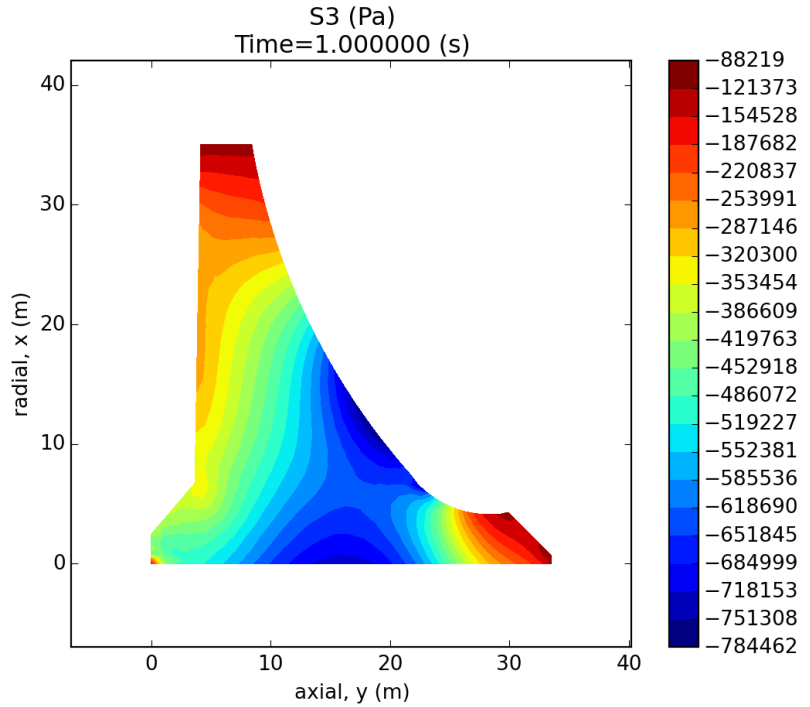
EXAMPLE: ANALYZING A DAM (BEETALOO DAM) PLANE STRAIN



Make part
Assign material



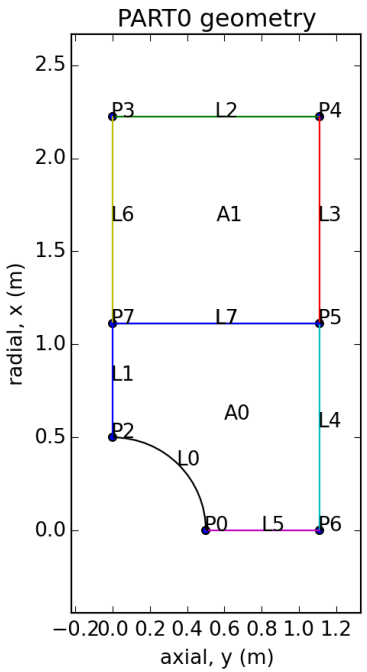
Mesh part
Apply constraints
Apply pressures
Apply gravity



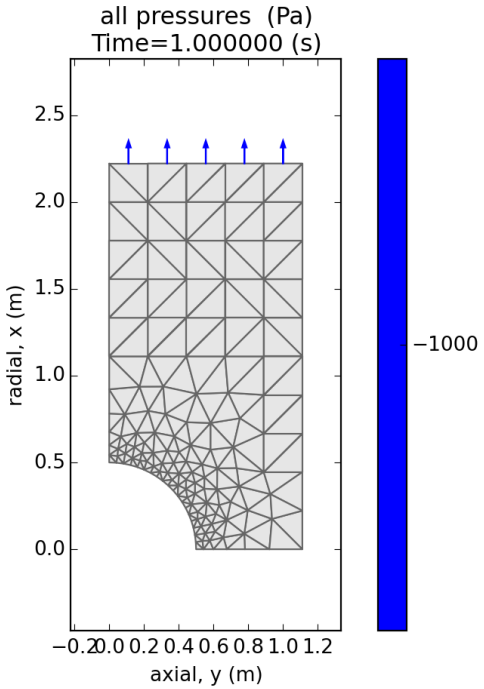
Solve model
View Results

File: hole_model.py

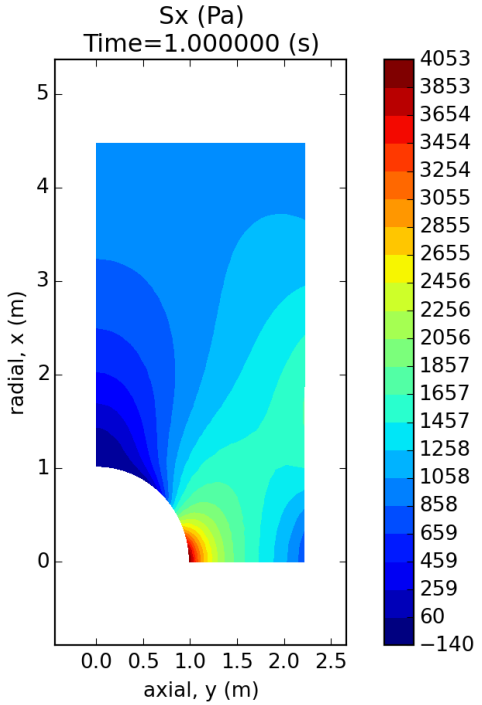
EXAMPLE: HOLE IN PLATE UNDER TENSION PLANE STRESS



Make part
Assign material



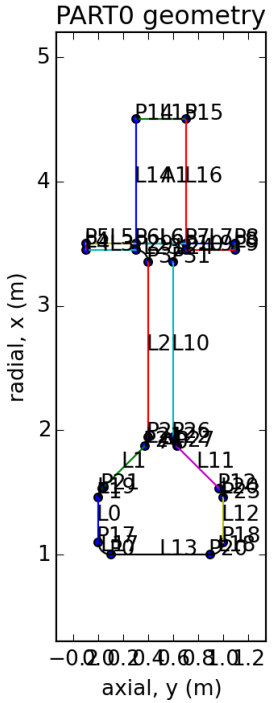
Mesh part
Apply constraints
Apply pressures



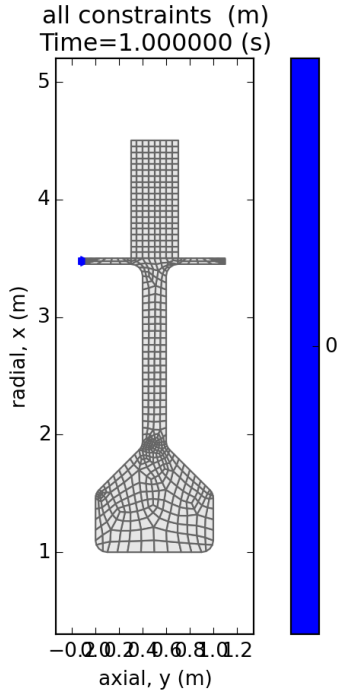
Solve model
View Results

File: compr_rotor_stage.py

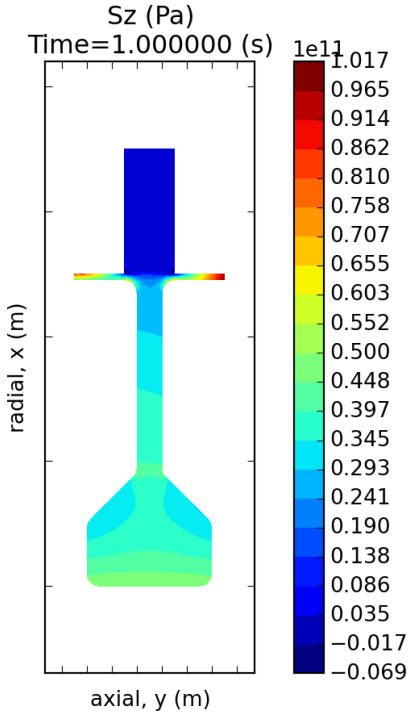
EXAMPLE: COMPRESSOR DISK OR TURBINE DISK AXISYMMETRIC



Make part
Assign material



Mesh part, set thickness on airfoil
Apply constraints
Apply speed

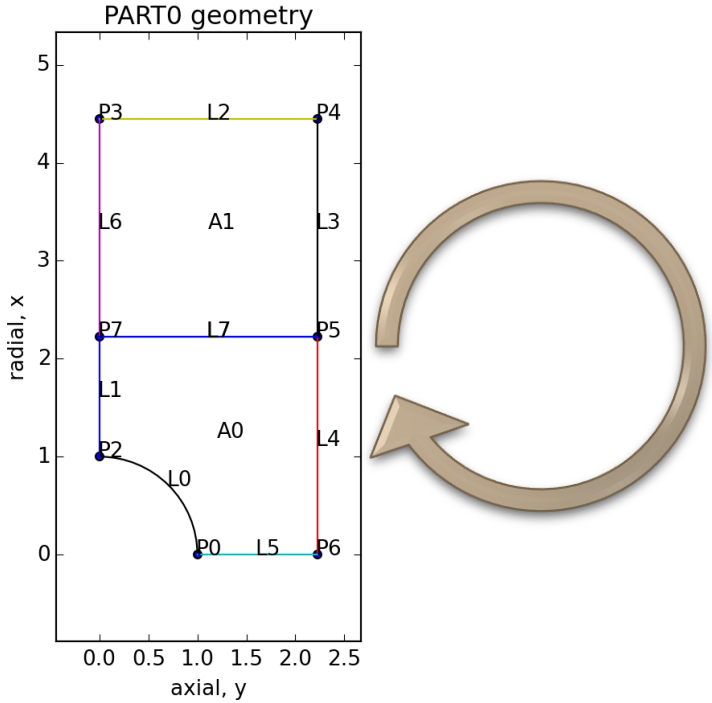


Solve model
View Results

File: hole_kt.py

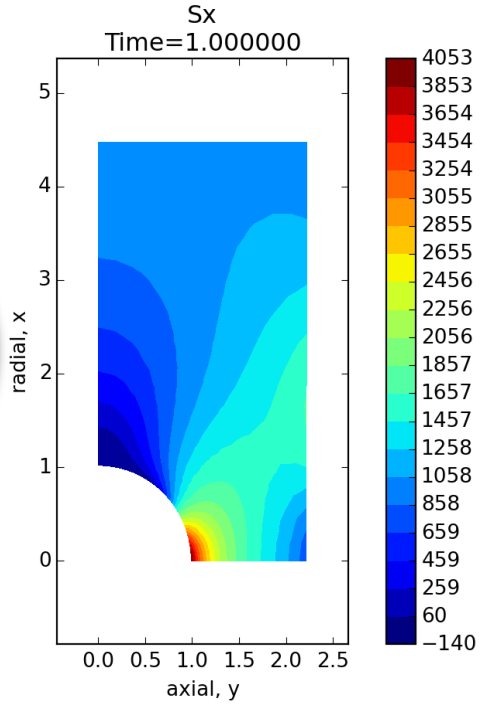
EXAMPLE: DESIGN STUDY PETERSON TENSION HOLE IN PLATE, PG 1

(meters)

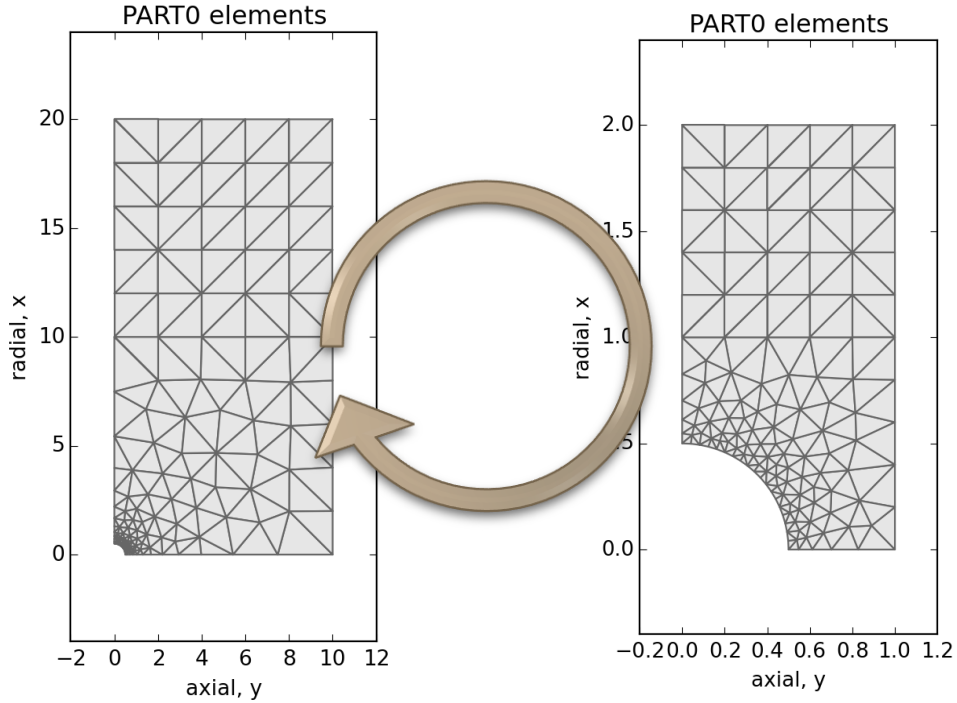


Make part
Assign material
Mesh Part
Apply constr + press

(Pa = N/(m^2))



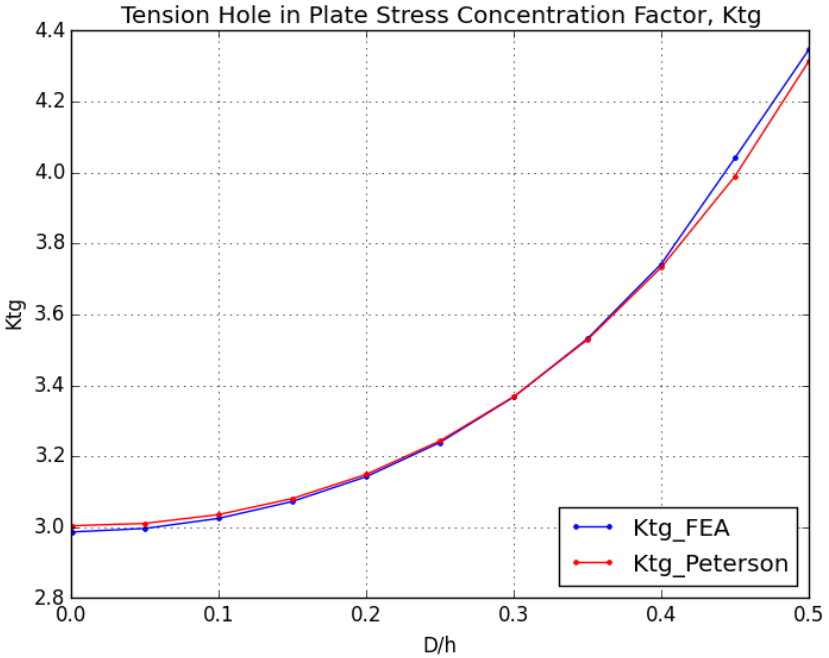
Solve model
Extract Kt

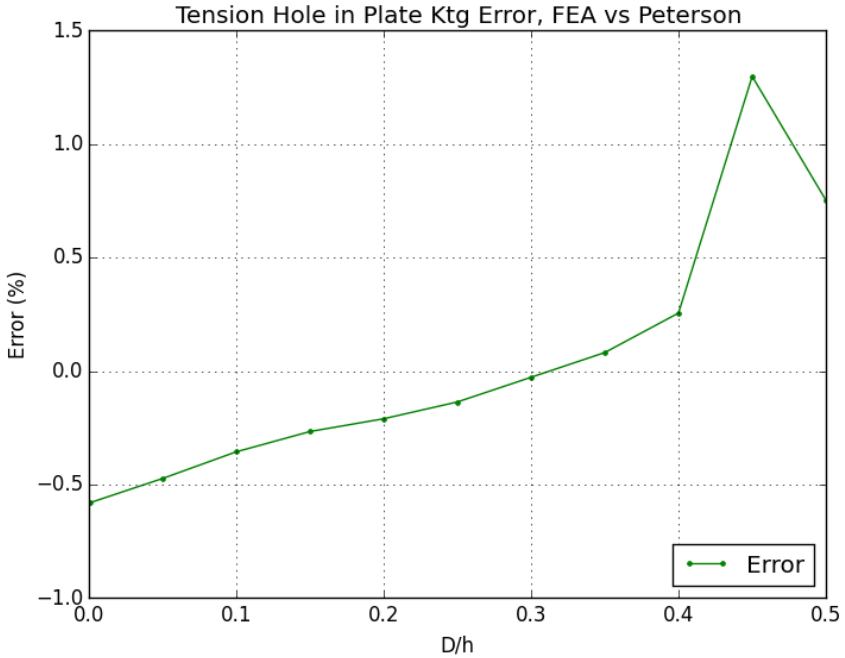


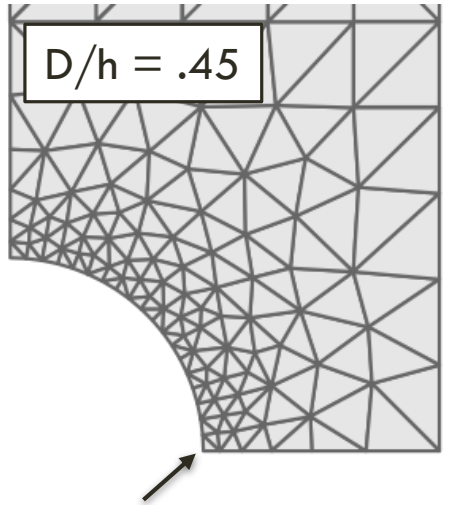
Run multiple models with a range of plate widths, using a constant hole size.
Compare Calculix FEA results with Peterson predicted results.

File: hole_kt.py

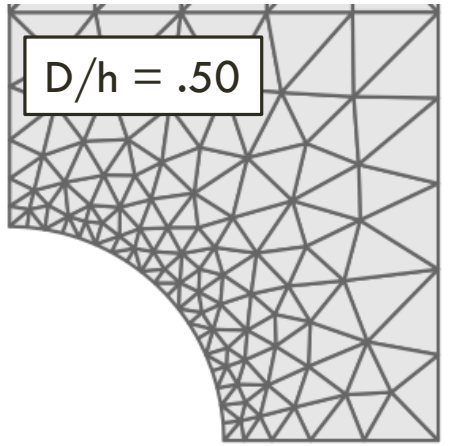
EXAMPLE: DESIGN STUDY PETERSON TENSION HOLE IN PLATE, PG 2







Error may be higher
Because only one element
on this corner
All other runs had 2 like below



Run multiple models with a range of plate widths, using a constant hole size.
Compare Calculix FEA results with Peterson predicted results.

Calculix FEA results are accurate to within 1.5% of Peterson's results. Error jump is probably due to layout of local elements.
19 elements used on arc, 2nd order tris used

File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG1

Import the pycalculix library and define a model

This model will hold all of our geometry, materials, loads, constraints, elements, and nodes.

```
1  from pycalculix import FeaModel
2
3  # Vertical hole in plate model, make model
4  proj_name = 'hole_model'
5  a = FeaModel(proj_name)
6  a.set_units('m')    # this sets dist units to meters, labels our consistent units
```

File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG2

Define the variables that we'll use to draw the part

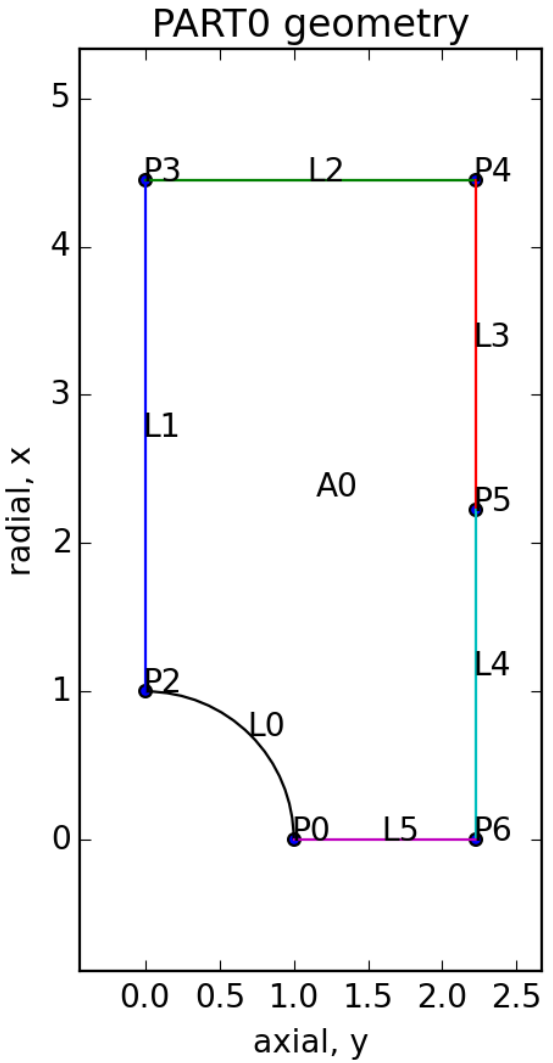
```
 8  # Define variables we'll use to draw part geometry
 9  diam = 2.0 # hole diam
10  ratio = 0.45
11  width = diam/ratio   #plate width
12  print('D=%f, H=%f, D/H=%f' % (diam, width, diam/width))
13  length = 2*width  #plate length
14  rad = diam/2   #hole radius
15  vdist = (length - 2*rad)/2  #derived dimension
16  adist = width/2             #derived dimension
```

File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG3

Draw the part. We have to make a PartMaker instance to store the part.
Part must be drawn in CLOCKWISE direction
x = vertical axis, also known as the 'radial' axis
y = horizontal axis, also known as the 'axial' axis
Draw_line_rad = draw radial line (vertical)
Draw_line_ax = draw axial line (horizontal)

```
18  # Draw part geometry, you must draw the part CLOCKWISE
19  # coordinates are x, y = radial, axial
20  b = a.PartMaker()
21  b.goto(0.0,rad)
22  b.draw_arc(rad, 0.0, 0.0, 0.0)
23  b.draw_line_rad(vdist)
24  b.draw_line_ax(adist)
25  b.draw_line_rad(-length/4.0)
26  b.draw_line_rad(-length/4.0)
27  b.draw_line_ax(-(adist-rad))
28  a.plot_geometry(proj_name+'_prechunk') # view the geometry
```



File: hole_model.py

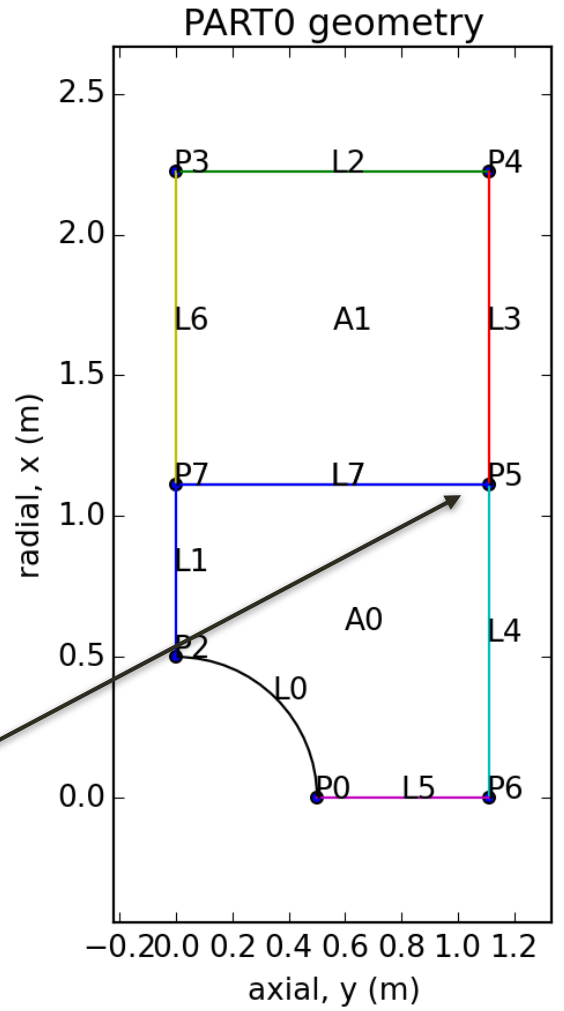
WALK THROUGH, HOLE IN PLATE, PG4

Chunking tells the program to try to cut the area into smaller pieces

It cuts the part at points. It draws a perpendicular line then cuts the part with it.

Chunking can help you make a better quality mesh.
It is required for CGX meshing, but not for GMSH meshing.

```
30  # Cut the part into easier to mesh areas
31  b.chunk() # cut the part into area pieces so CGX can mesh it
32  a.plot_geometry(proj_name+'_chunked') # view the geometry
```



Area was chunked at P5

File: hole_model.py

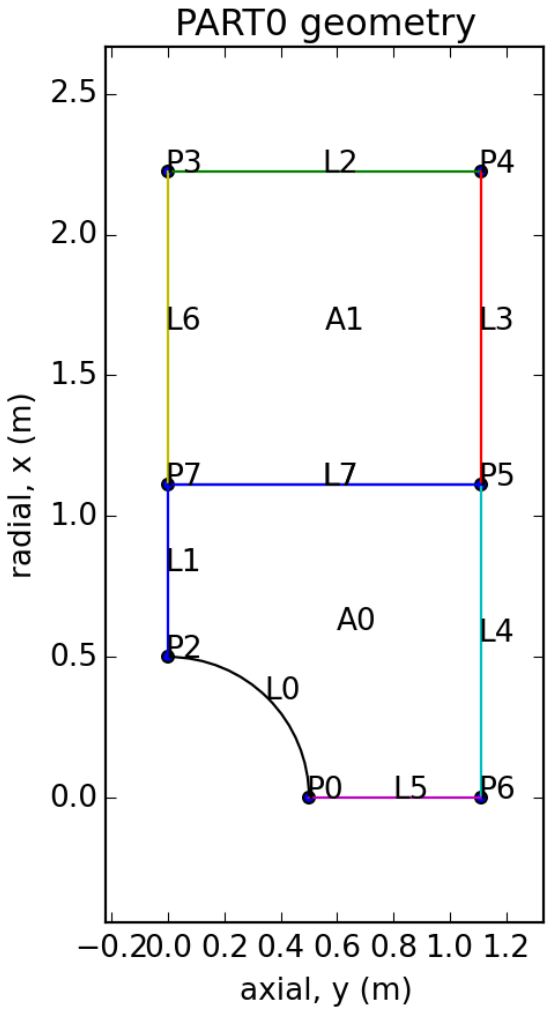
WALK THROUGH, HOLE IN PLATE, PG5

Sets the loads and constraints

Positive pressures push on the part. Negative pressures pull on the part.

Note: we can do this either before or after meshing because the program stores loads on geometry (points, lines, areas) rather than the mesh.

```
34  # set loads and constraints
35  a.set_load('press',b.top,-1000)
36  a.set_constr('fix',b.left,'y')
37  a.set_constr('fix',b.bottom,'x')
```

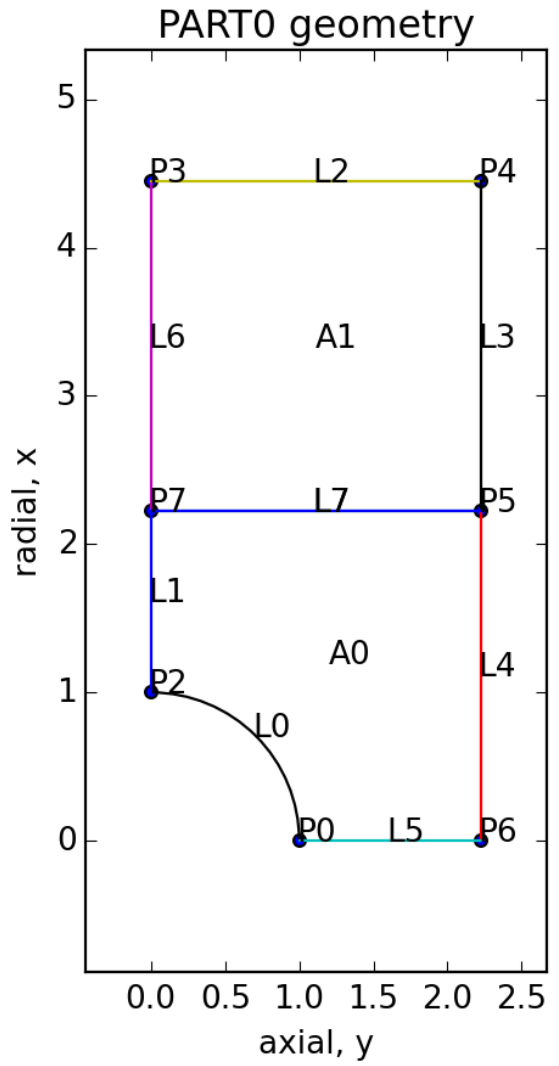


File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG6

Set the part material

```
39  # set part material
40  mat = a.MatlMaker('steel')
41  mat.set_mech_props(7800, 210*(10**9), 0.3)
42  a.set_matl(mat, b)
```



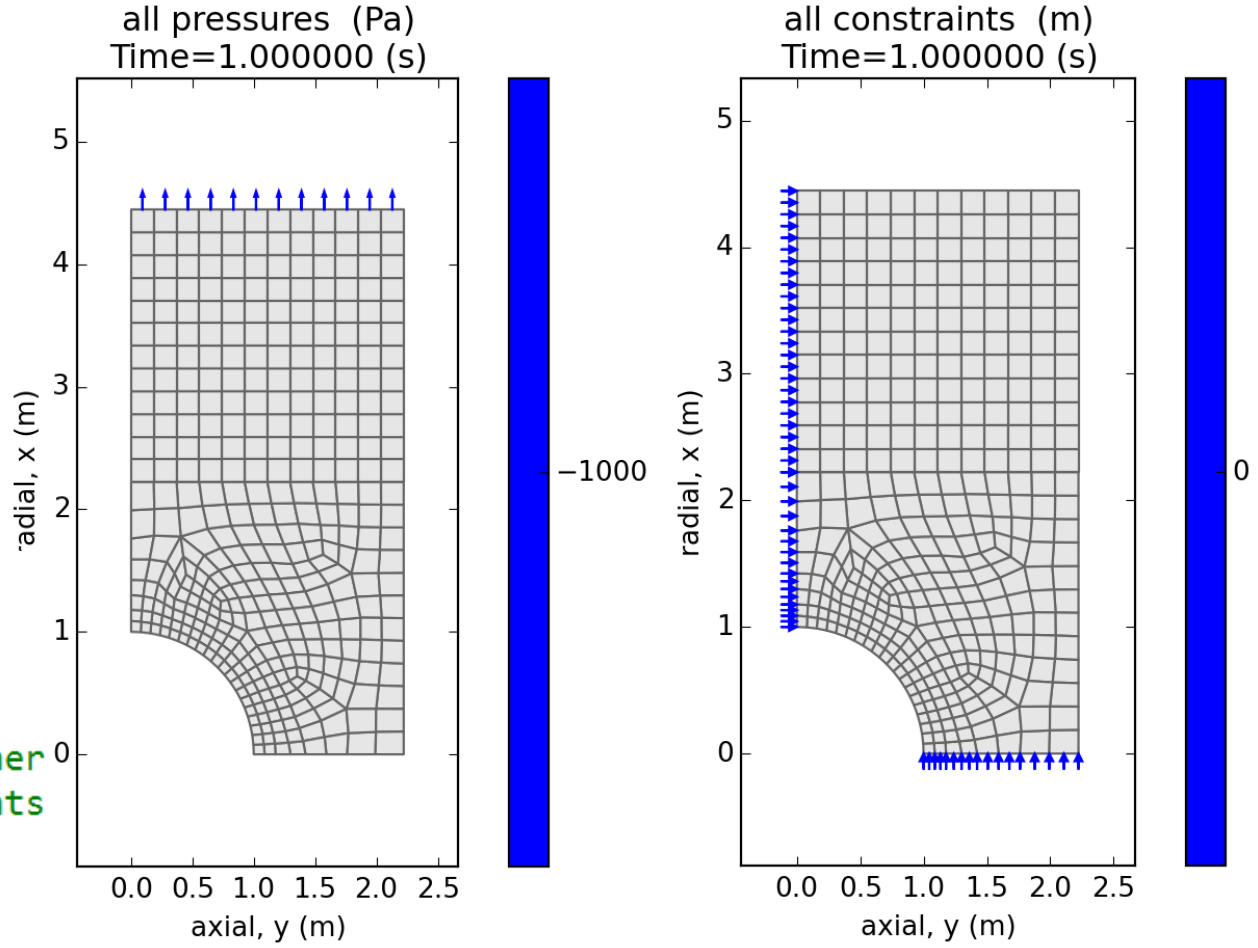
File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG7

Mesh the part

set_eshape(shape='quad' or 'tri', order=1 or 2)
set_etype(part, etype, thickness)
etype:
'plstress' = plane stress, thickness is required
'plstrain' = plane strain, thickness is required
'axisym' = axisymmetric, thickness is not required

```
44  # set the element type and mesh database
45  a.set_eshape('quad', 2)
46  a.set_etype(b, 'plstress', 0.1)
47  a.get_item('L0').set_ediv(20) # set element divisions
48  a.mesh(1.0, 'gmsh') # mesh 1.0 fineness, smaller is finer
49  a.plot_elements(proj_name+'_elem')   # plot part elements
50  a.plot_pressures(proj_name+'_press')
51  a.plot_constraints(proj_name+'_constr')
```



File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG8

Make and solve the model.

Python console output on the right.

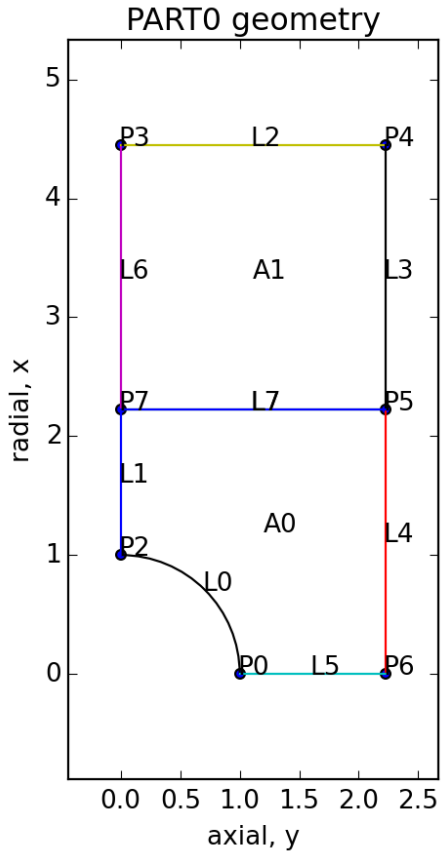
```
53  # make and solve the model
54  mod = a.ModelMaker(b, 'struct')
55  mod.solve()
```

```
Solving done!
Reading results file: hole_model.frd
Reading nodes
Reading displ storing: ux,uy,uz,utot
Reading stress storing: Sx,Sy,Sz,Sxy,Syz,Szx,Seqv,S1,S2,S3
Reading strain storing: ex,ey,ez,exy,eyz,ezx,eeqv
Reading force storing: fx,fy,fz
The following times have been read:  [1.0]
Done reading file: hole_model.frd
Results file time set to: 1.000000
```

File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG9

Query our results. Check the max stress and the reaction forces.




```
57  # view and query results
58  sx = mod.rfile.get_nmax('Sx')
59  print('Sx_max: %f' % (sx))
60  [fx, fy, fz] = mod.rfile.get_fsum(b.get_item('L5'))
61  print('Reaction forces (fx,fy,fz) = (%12.10f, %12.10f, %12.10f)' % (fx, fy, fz))
```

File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG10

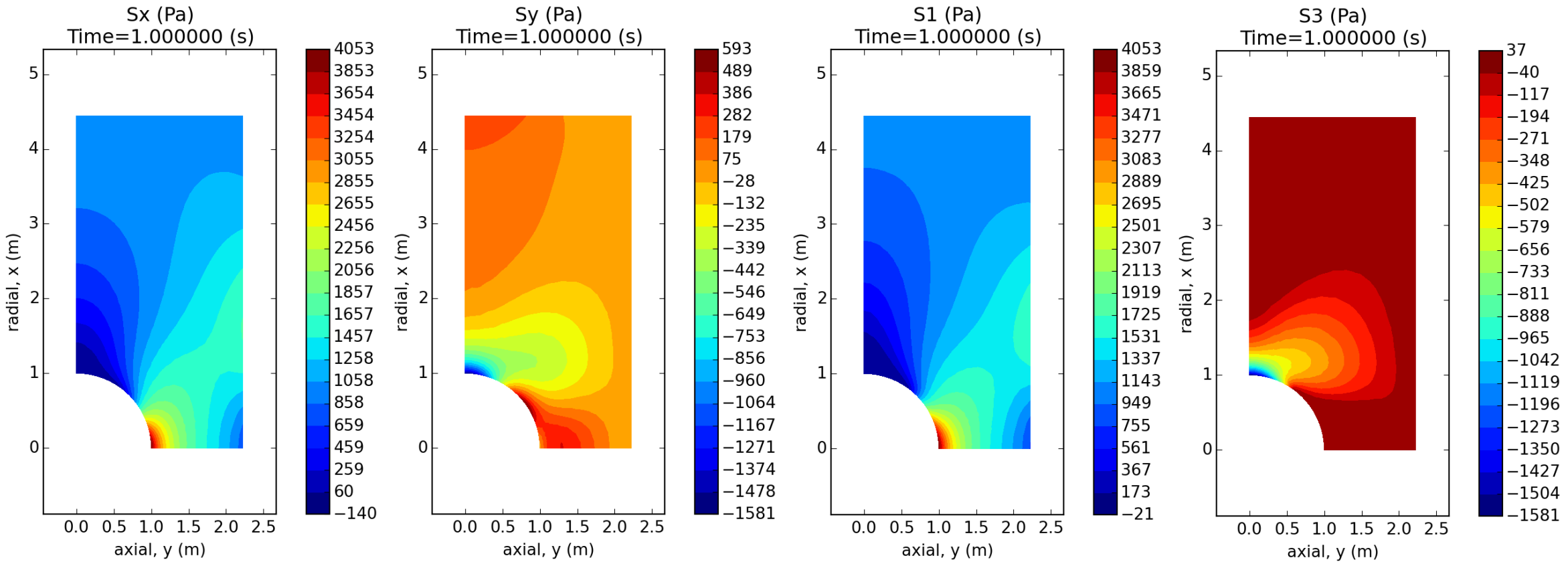
Plot our results.

Interactive plotting is suppressed with the display variable, but files are saved.

```
63  # Plot results
64  disp = False
65  fields = 'Sx,Sy,S1,S2,S3,Seqv,ux,uy,utot'    # store the fields to plot
66  fields = fields.split(',')
67  for field in fields:
68      fname = proj_name+'_'+field
69      mod.rfile.nplot(field, fname, display=disp)
```

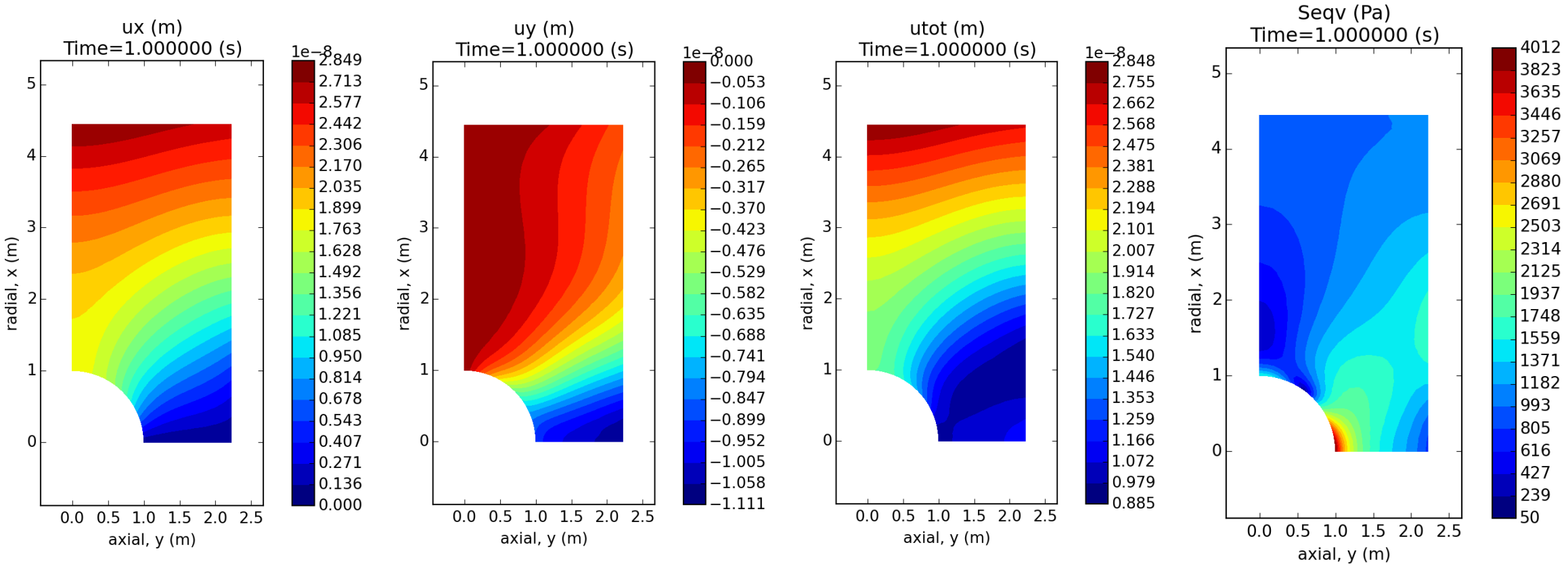
File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG11, PLOTS



File: hole_model.py

WALK THROUGH, HOLE IN PLATE, PG12, PLOTS



FUTURE WORK

Add element results plotting

Make lists for lines and signed lines (need to write a signed lines class)

Add struct-thermal and thermal support

Auto-detect contact regions between parts

Add compression supports

CAD import of brep and igs via gmsh

CAD export via gmsh

Bolted joint example perhaps, nodal thickness on bolt and nut areas

CONCLUSION

It's now possible to build geometry, mesh it, apply loads and constraints, and plot and query results all in one place.

One can now do design studies very easily with this tool.

Suggestions?

Feedback?

Please let me know at justin.a.black@gmail.com